实验四

**Python程序性能评测**

**实验目的**

1、利用python采集与分析cpu,memory,disk的信息;

2、python performance基准测试

**实验内容**

1、python获取机器的常用配置信息

2、python获取程序执行时间

**实验步骤和结果**

**实验一 python获取机器的常用配置信息**

[实验内容]

用python的psutil模块输出cpu,memory,disk信息

import psutil as ps

​

metrics = {}

​

#cpu

cpu = ps.cpu\_times()

metrics['cpu\_usage'] = ps.cpu\_percent(interval = 1)

metrics['cpu\_user'] = cpu.user

metrics['cpu\_system'] = cpu.system

metrics['cpu\_idle'] = cpu.idle

​

#Memory

mem = ps.virtual\_memory()

metrics['mem\_free'] = mem.free

metrics['mem\_used'] = mem.used

metrics['mem\_available'] = mem.available

metrics['mem\_usage'] = mem.percent

​

​

#Disk

disk = ps.disk\_io\_counters(perdisk=True)

for prefix in disk.keys():

metrics[prefix + '\_' + 'read\_count'] = disk[prefix].read\_count

metrics[prefix + '\_' + 'write\_count'] = disk[prefix].write\_count

metrics[prefix + '\_' + 'read\_bytes'] = disk[prefix].read\_bytes

metrics[prefix + '\_' + 'read\_bytes'] = disk[prefix].write\_bytes

​

​

for item in metrics.items():

print(item)

​



[实验结果]

('cpu\_usage', 3.3)

('cpu\_user', 13092.58)

('cpu\_system', 3097.12)

('cpu\_idle', 672095.85)

('mem\_free', 3361009664)

('mem\_used', 3225280512)

('mem\_available', 3878764544)

('mem\_usage', 52.9)

('sda1\_read\_count', 46)

('sda1\_write\_count', 0)

('sda1\_read\_bytes', 0)

('sda2\_read\_count', 42)

('sda2\_write\_count', 0)

('sda2\_read\_bytes', 0)

('sda3\_read\_count', 2)

('sda3\_write\_count', 0)

('sda3\_read\_bytes', 0)

('sda5\_read\_count', 45)

('sda5\_write\_count', 0)

('sda5\_read\_bytes', 0)

('sda6\_read\_count', 42)

('sda6\_write\_count', 0)

('sda6\_read\_bytes', 0)

('sda7\_read\_count', 161)

('sda7\_write\_count', 0)

('sda7\_read\_bytes', 0)

('sda8\_read\_count', 297742)

('sda8\_write\_count', 478514)

('sda8\_read\_bytes', 12095332352)

('sda9\_read\_count', 20881)

('sda9\_write\_count', 3346)

('sda9\_read\_bytes', 1125842944)

('sr0\_read\_count', 0)

('sr0\_write\_count', 0)

('sr0\_read\_bytes', 0)

获取的本地信息，每个人结果一般不同

**实验二 python获取程序执行时间**

[实验内容]

四种替换算法时间性能比较

orignal\_str = "Profiling a Python program is doing a dynamic analysis"\

"that measures the execution time of the program and"\

"everything that compose it."

exec\_times = 100000 #执行次数

​

#四种替换算法

def slowest\_replace():

replace\_list = []

for i, char in enumerate(orignal\_str):

c = char if char != " " else "-"

replace\_list.append(c)

return "".join(replace\_list)

​

​

def slow\_replace():

replace\_str = ""

for i, char in enumerate(orignal\_str):

c = char if char != " " else "-"

replace\_str += c

return replace\_str

​

​

def fast\_replace():

return "-".join(orignal\_str.split())

​

​

def fastest\_replace():

return orignal\_str.replace(" ", "-")

​

#手动设置时间断点分析函数执行时间

def \_time\_analyze\_(func):

from time import clock

start = clock()

for i in range(exec\_times):

func()

finish = clock()

print (func.\_\_name\_\_ , ":", finish - start,'s')

​

​

def simple\_profile():

print ("\*" \* 40, "\nSimple time analyze")

for fun in [slowest\_replace, slow\_replace, fast\_replace, fastest\_replace]:

\_time\_analyze\_(fun)

​

#用python的timeit模块分析函数执行时间

def \_timeit\_analyze\_(func):

from timeit import Timer

t1 = Timer("%s()" % func.\_\_name\_\_, "from \_\_main\_\_ import %s" % func.\_\_name\_\_)

print (func.\_\_name\_\_ , ":", t1.timeit(exec\_times))

​

​

def timeit\_profile():

print ("\*" \* 40, "\nModule timeit analyze")

for fun in [slowest\_replace, slow\_replace, fast\_replace, fastest\_replace]:

\_timeit\_analyze\_(fun)

​

​

if \_\_name\_\_ == "\_\_main\_\_":

simple\_profile()

timeit\_profile()



[实验结果]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Simple time analyze

slowest\_replace : 1.5494550000000018 s

slow\_replace : 1.2513919999999992 s

fast\_replace : 0.14179299999999984 s

fastest\_replace : 0.030080999999999136 s

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Module timeit analyze

slowest\_replace : 1.558246290020179

slow\_replace : 1.2700494629971217

fast\_replace : 0.14182965198415332

fastest\_replace : 0.030810013005975634

**实验三 python程序可视化分析**

[实验内容]

我选择一个开源的性能分析工具vprof，原因是它可以很方便地生成火焰图。火焰图是类似下面的图形，具体了解火焰图，可以访问《如何读懂火焰图》这篇文章。

http://www.ruanyifeng.com/blog/2017/09/flame-graph.html

安装vprof很简单，使用下面这个命令：

pip install vprof



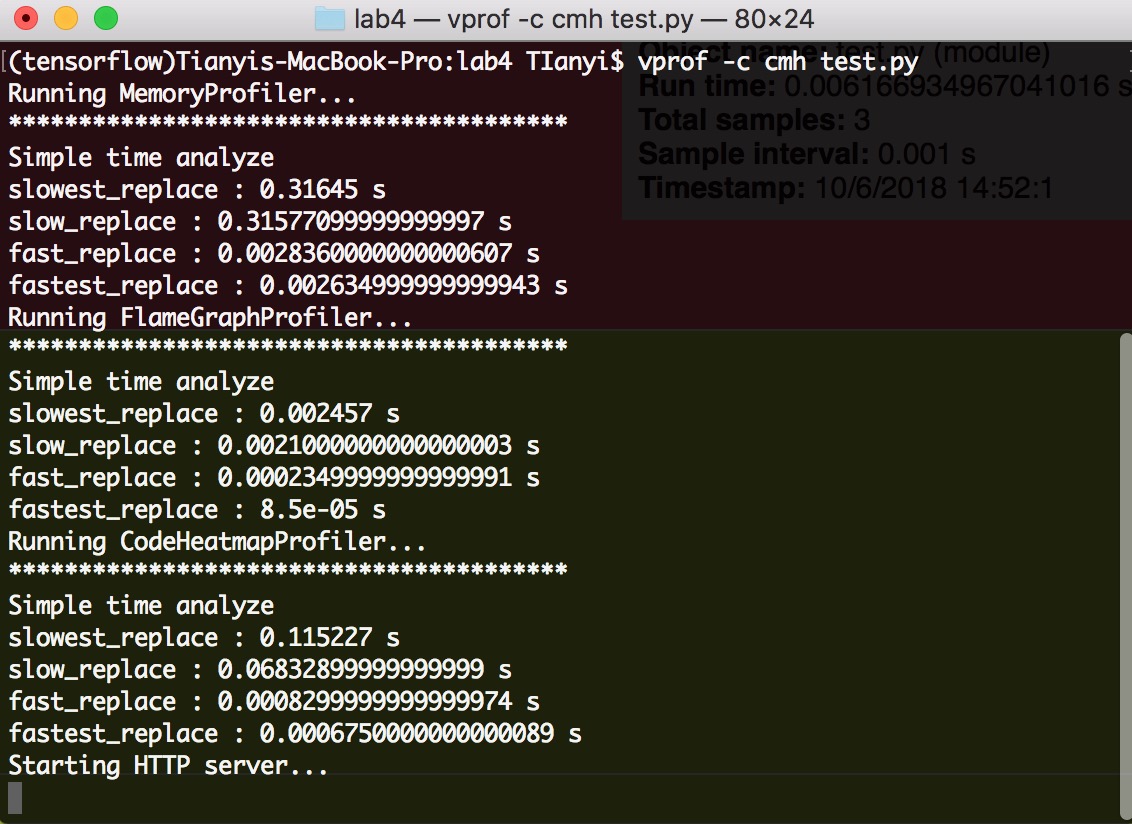
收集性能数据可以使用如下命令：

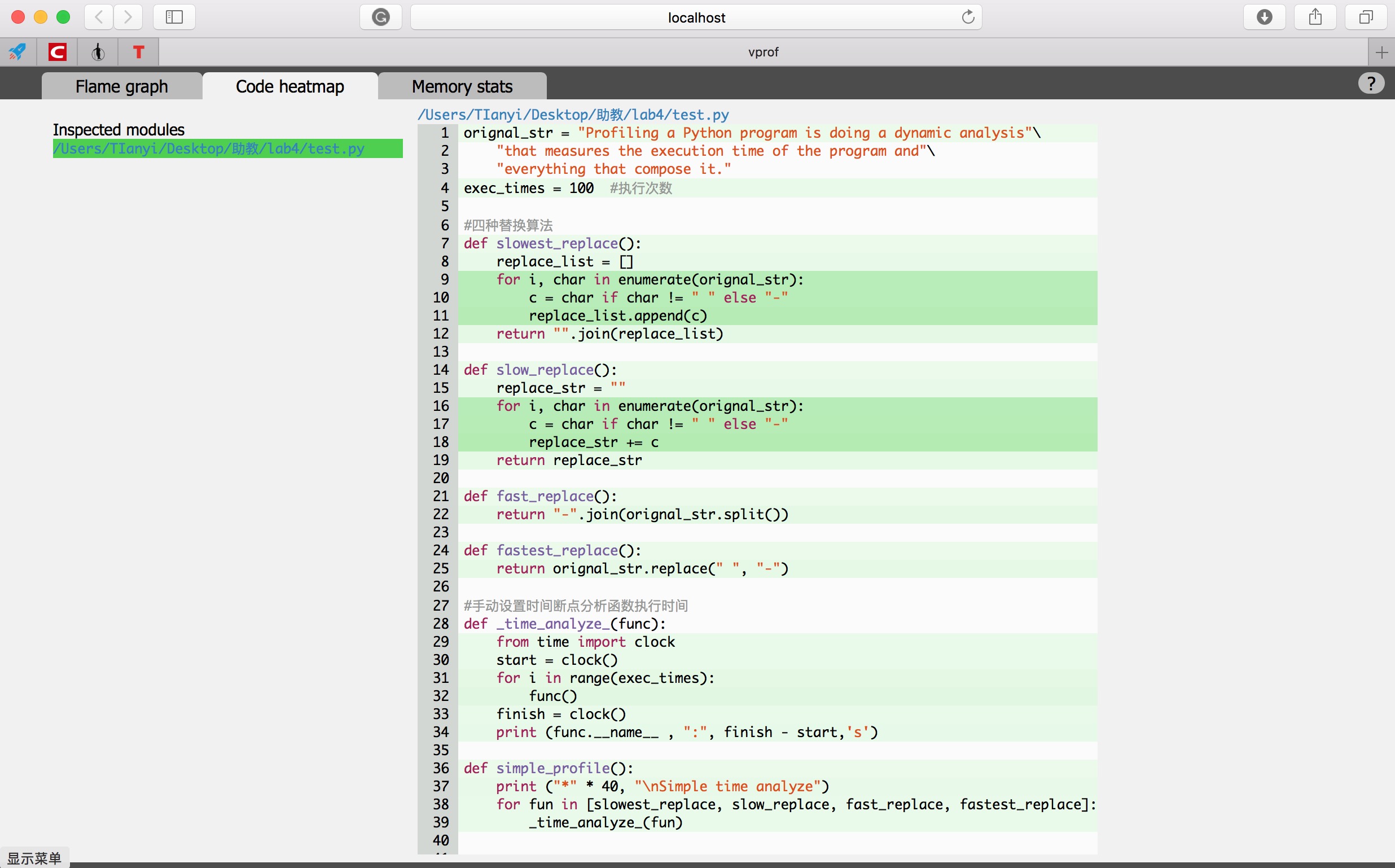
vprof -c cmh test.py



这样的命令，其中-c代表配置。配置里面的c，表示cpu火焰图，m表示内存图，h表示代码热图。当主程序退出后，vprof会自动收集这些数据，并且启动一个http服务器，自动打开浏览器将指定的图表打开展示出来。

保存实验二的代码为test.py，使用vprof运行代码效果如下：

​​

​​

**[作业]**

1、比较常用排序算法（冒泡排序、选择排序、插入排序，归并排序）的时间性能。

提示：

（1）python 中可以引入time模块，使用time.time()获取当前计算机时间。

（2）归并排序代码直接给出，其他三种排序自行完成。

def merge\_sort(nums):

lens = len(nums)

count = lens

n = 2

def sort\_two\_list(list1,list2):

len1 = len(list1)

len2 = len(list2)

list3 = []

i,j=0,0

while i<len1 and j<len2:

if list1[i] < list2[j]:

list3.append(list1[i])

i += 1

else:

list3.append(list2[j])

j += 1

list3 += list1[i:]

list3 += list2[j:]

return list3

for i in range(0,lens-1,2):

if nums[i] > nums[i+1]:

nums[i],nums[i+1] = nums[i+1],nums[i]

while n<=lens:

list3 = []

for j in range(0,lens,2\*n):

list3 += sort\_two\_list(nums[j:j+n],nums[j+n:j+2\*n])

nums = list3

n += n

return nums



（3）比较排序算法需要使用较长的无序数组，尝试使用100，1000，10000三种不同长度的无序数组进行测试，输出每次测试不同排序算法的运行时间。

**[做作业说明]**

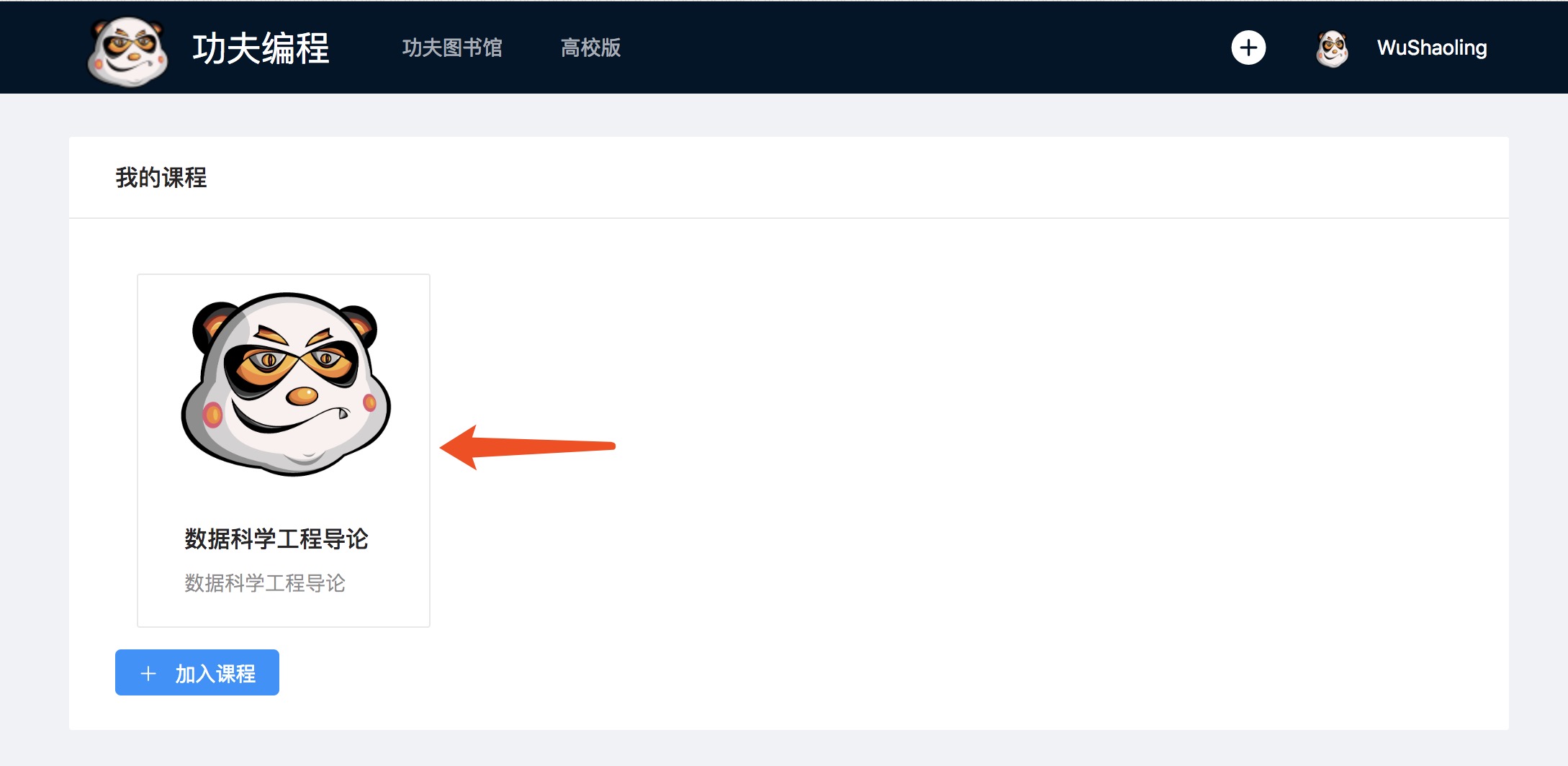
1.打开功夫编程首页，点击高校版按钮进入功夫编程高校版(未登录的用户需先登录)

​​

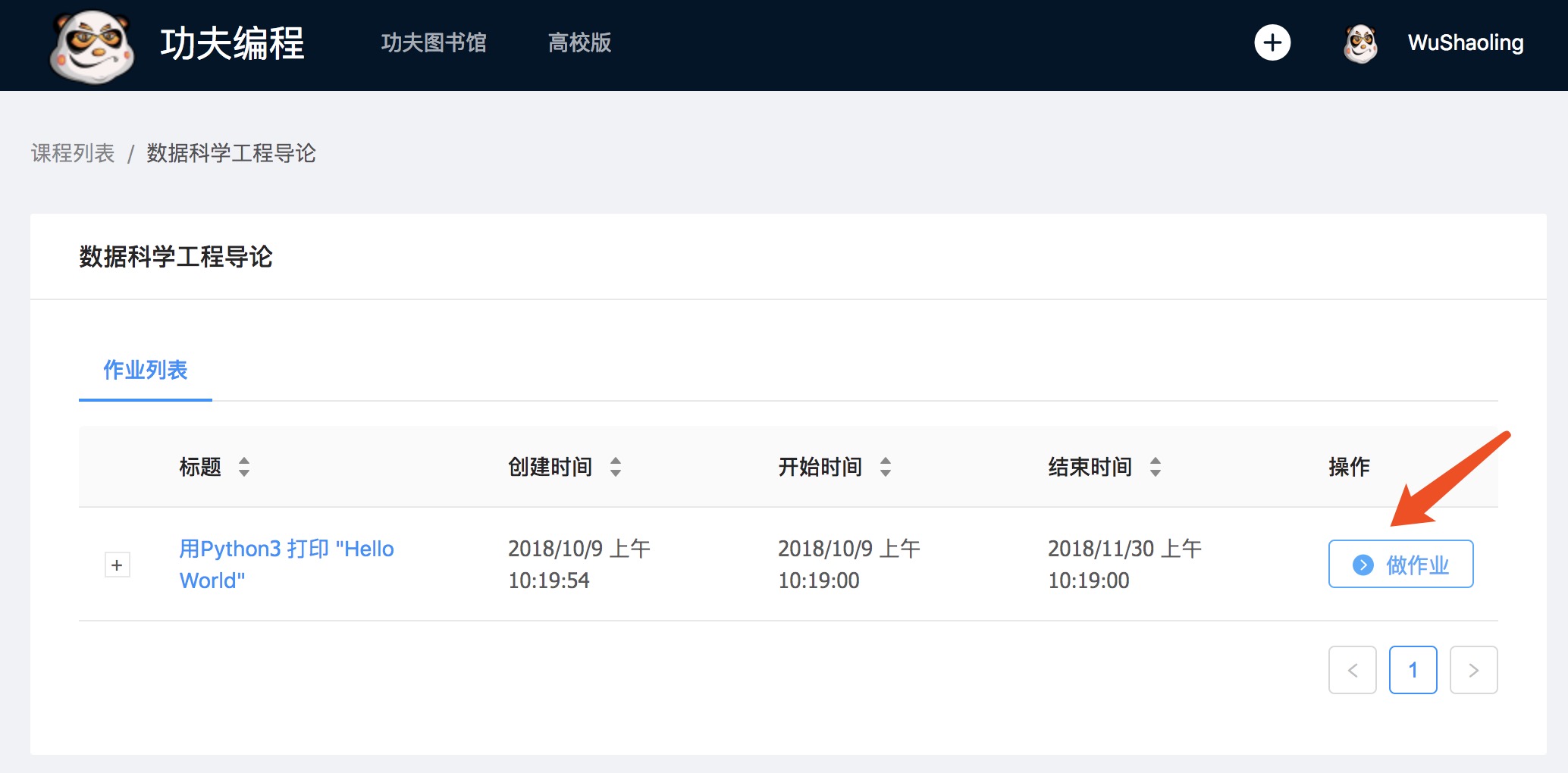
2.未加入相应课程的则首先需要加入课程。在高校版页面中，点击加入课程按钮，输入教师提供的课程代码加入课程，课程代码为2swf8c。

​​

3.加入课程以后，点击相应的课程，进入作业列表页面。

​​

4.在作业列表页面中，选择指定的作业并点击做作业按钮，进入workspace做作业页面

​​

5.在workspace页面中，编辑作业，作业完成以后保存即可

​